DETAILED ACTION

1. This Office Action is responsive to amendments filed in application No. 10/692,004 on November 17, 2009. Claims 1-20 are pending and have been examined.

Claim Rejections – 35 USC § 103

2. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

The factual inquiries set forth in *Graham* **v.** *John Deere Co.*, 383 U.S. 1, 148 USPQ 459 (1966), that are applied for establishing a background for determining obviousness under 35 U.S.C. 103(a) are summarized as follows:

1. Determining the scope and contents of the prior art.
2. Ascertaining the differences between the prior art and the claims at issue.
3. Resolving the level of ordinary skill in the pertinent art.
4. Considering objective evidence present in the application indicating obviousness or nonobviousness.

3. **Claims 1-20** rejected under 35 U.S.C. 103(a) as being unpatentable over Applicant's admitted prior art (AAPA) in view of Price et al. (US 2002/0120607 A1) and Vermeire et al. (US 2001/0025372 A1)

AAPA teaches in Claim 1:

A process for transferring pen data between unmanaged and managed code on a computing device for use by an application on the computing device, the unmanaged code being code native to and executed directly by a processor of the computing device **(Figure 3 shows the unmanaged and managed sections as the data is converted and also shows the end result is used in runtime callable wrapper 305/307 and the applications used within)**, the managed code being executed in a common language run-time environment of a framework operating on the computing device **(Figure 3, [0009] shows the common language run-time to manage the various types of codes, such as managed and unmanaged code)**, the common language run-time environment of the framework executing the managed code independent of a type of the processor of the computing device **(Figure 3, note the managed code is a separate section of code, apart from the raw, unmanaged code of the stylus input)**, the process comprising the steps of:

receiving pen data in a service operating in a user mode **(Please note in Figure 3 of a kernel section being separated from 302. Programming windows often consists of a kernal mode and a user mode/windows programming mode which are distinct from each other. Examiner asserts Official Notice to this common architecture. The separated area is indeed the user mode area given the application area for user input)** on the computing device written in unmanaged code and separate from the application on the computing device **(Figure 3, [0009] shows the component 301, on the unmanaged side)**, the pen data being generated by a digitizer of the computing device upon movement of a stylus with respect to a surface of the digitizer and being received by the pen service from a pen device driver operating in a kernal mode on the computing device, the pen data including at least one location on the digitizer of the

stylus **(Figure 3, [0009] shows the pen device drivers 301 which receives information from a digitizer. Please note that 301 is in the kernel mode)**;

the pen service transferring information related to said pen data to a memory on the computing device designated to be shared between unmanaged code and managed code, **(Figure 3, [0011] discloses the use of a memory for storing the pen information and then converting that data. Please see the combination below)**;

loading a pen input component into the application on the computing device, the pen input component being written in unmanaged code and being designated to communicate with the pen service for the application **(Figure 3, [0009] shows component 302 which is on the unmanaged side and communicates with 301 to send the information to 305)**;

the loaded pen input component further transferring to a stylus input subsystem of the application on the computing device, the stylus input subsystem being separate from the pen input component and being written in managed code and executed in the common language runtime environment **(Figure 3, [0009], this is done in the runtime callable wrapper 305 which contains a type library importer. [0009] discloses 305 and 306 import the data from the pen input component (read these as being a stylus input subsystem, which is reasonable given a subsystem is just a follow-up to the primary pen component). [0009] discloses command that can be invoked to retrieve information from pen component 302 and the callable wrapper can obviously, if not inherently, perform other actions to convert the unmanaged code)**;

the stylus input subsystem of the application receiving the pen input component **(Figure 3, [0009] show the pen component's information is retrieved by 305 which invokes a command to get the data)**; and

the stylus input subsystem of the application submitting with a retrieval command to the shared memory to retrieve said information from said shared memory, the retrieval command as submitted by the stylus input subsystem comprising a P-invoke method "GetData" which takes into account a context of the stylus input subsystem. **(Please note [0009] and [0076] for example. Both detail marshalling code and converting code using 304, 305 and 306. It is also obvious and well known that GetData is a commonly used command in CLR and Examiner asserts Official Notice to this. In addition, it is obvious, if not inherent, that there is a command to initiate the retrieval of data in Figure 3)**; but

AAPA does not explicitly teach of a "mutual exclusion" shared memory so that non-simultaneous sharing of code can be achieved between the unmanaged and managed sections.

However, the use of mutex is well known in the art.

To emphasize, Price teaches of using a mutex to allow multiple programs to share the same resource, but not simultaneously, (Price, [0060]).

Therefore, it would be obvious to one of ordinary skill in the art at the time of the invention to integrate the mutex, as taught by Price, with the AAPA's device with the motivation that by non-

simultaneously sharing the mutex, sequential operation can be done and errors can be prevented from different programs trying to access the same object at the same time.

AAPA also does not explicitly teach of a pointer to reference different parts of the memory, pen services, pen input component, etc, to either to store or retrieve them using the pointers. In addition, they are used in a P-invoke-style interface to interface between the unmanaged and managed portions.. However, both of these elements are well known in the art.

To emphasize, in the same field of endeavor, framework systems, Vermeire teaches of using memory addresses or pointers in the runtime framework as a means to access data, (Vermeire, [0033]). As for the P-invoke interfacing, this is well known in the programming field and examiner asserts Official Notice to the common use of such an interfacing in this field of endeavor and point out KSR principles which provide rationale on why one of skill would use the teaching. These include well known technique, simple substitution and obvious to try given its common use and benefits.

Therefore it would be obvious to one of ordinary skill in the art at the time of the invention to integrate the use of pointers, as taught by Vermeire, with the AAPA with the motivation that it is common to use pointers in the art as well as for better efficiency, memory management and to prevent memory leaks.

AAPA, Price and Vermeire teach in Claim 2:

The process according to claim 1, further comprising the steps of:

transferring additional information from said at least in part managed application to said shared memory **([0011] discloses a memory. Price has been combined with AAPA to teach of a mutex as well)**;

transferring a pointer that points to said additional information to said component; retrieving said additional information from said shared memory. **(The combination with Vermeire teaches to use pointers to access and store data in memory)**

AAPA teaches in Claim 4:

The process according to claim 1, further comprising the step of: exchanging information through a COM interface. **(Figure 3, [0009] shows the COM 303)**

AAPA teaches in Claim 5:

The process according to claim 1, said component being a pen services component. **(Figure 3, [0009] discloses the component 302 may include pen services)**

AAPA teaches in Claim 6:

The process according to claim 1, said application including a pen input managed client. **(This is obvious in light of Figure 3's unmanaged and managed code relating to pen input from a digitizer)**

AAPA teaches in Claim 7:

The process according to claim 1, said component receiving input from at least one pen device driver. **(Figure 3, [0009] shows the pen device drivers 301)**

AAPA teaches in Claim 8:

A system for transferring information between unmanaged code and managed code on a computing device, the unmanaged code being code native to and executed directly by a processor of the computing device for use by an application on the computing device **(Figure 3 shows the unmanaged and managed sections as the data is converted and also shows the end result is used in runtime callable wrapper 305/307 and the applications used within)**, the managed code being executed in a common language run-time environment of a framework operating on the computing device **(Figure 3, [0009] shows the common language run-time to manage the various types of codes, such as managed and unmanaged code)**, the common language run-time environment of the framework executing the managed code independent of a type of the processor of the computing device **(Figure 3, note the managed code is a separate section of code, apart from the raw, unmanaged code of the stylus input)**, the system comprising:

a shared memory on the computing device designated to be shared between unmanaged code and managed code **(Figure 3, [0011] discloses the use of a memory for storing the pen information and then converting that data)**;

a pen service operating in a user mode **(Please note in Figure 3 of a kernel section being separated from 302. Programming windows often consists of a kernal mode and a user mode/windows programming mode which are distinct from each other. Examiner**

asserts Official Notice to this common architecture. The separated area is indeed the user mode area given the application area for user input) on the computing device, the pen service being written in unmanaged code and receiving pen data **(Figure 3, [0009] shows the component 301, on the unmanaged side)**, the pen data being generated by a digitizer of the computing device upon movement of a stylus with respect to a surface of the digitizer and being received by the pen service from a pen device driver operating in a kernel mode on the computing device, the pen data including at least one location on the digitizer of the stylus **(Figure 3, [0009] shows the pen device drivers 301 which receives information from a digitizer. Please note that 301 is in the kernel mode)**,

a pen input component loaded into the application on the computing device, the pen input component being written in unmanaged code and being designated to communicate with the pen service for the application **(Figure 3, [0009] shows component 302 which is on the unmanaged side and communicates with 301 to send the information to 305)**;

the pen input component further transferring information EDEto a stylus input subsystem of the application on the computing device, the stylus input subsystem being separate from the pen input component and being managed code and executed in the common language run-time environment **(Figure 3, [0009] shows the application which receives the managed code after it has been converted from the unmanaged code in the CLR. This is done in the runtime callable wrapper 305 which contains a type library importer. [0009] discloses 305 and 306 import the data from the pen input component (read these as being a stylus input subsystem, which is reasonable given a subsystem is just a follow-up to the primary pen component). [0009] discloses command that can be invoked to retrieve information from**

pen component 302 and the callable wrapper can obviously, if not inherently, perform other actions to convert the unmanaged code);

said stylus input subsystem having managed code receiving the transferred pointer from the pen input component and submitting the receiving pointer with a retrieval command to the shared memory to retrieve said information from said shared memory by way of the transferred pointer, the retrieval command as submitted by the stylus input subsystem comprising a P-invoke method "GetData", which takes into account a context of the stylus input subsystem **(Please note [0009] and [0076] for example. Both detail marshalling code and converting code using 304, 305 and 306. It is also obvious and well known that GetData is a commonly used command in CLR and Examiner asserts Official Notice to this. In addition, it is obvious, if not inherent, that there is a command to initiate the retrieval of data in Figure 3)**; but

AAPA does not explicitly teach of a "mutual exclusion" shared memory so that non-simultaneous sharing of code can be achieved between the unmanaged and managed sections.

However, the use of mutex is well known in the art.

To emphasize, Price teaches of using a mutex to allow multiple programs to share the same resource, but not simultaneously, (Price, [0060]).

Therefore, it would be obvious to one of ordinary skill in the art at the time of the invention to integrate the mutex, as taught by Price, with the AAPA's device with the motivation that by non-

simultaneously sharing the mutex, sequential operation can be done and errors can be prevented from different programs trying to access the same object at the same time.

AAPA also does not explicitly teach of a pointer to reference different parts of the memory, pen services, pen input component, etc, to either to store or retrieve them using the pointers. In addition, they are used in a P-invoke-style interface to interface between the unmanaged and managed portions.. However, both of these elements are well known in the art.

To emphasize, in the same field of endeavor, framework systems, Vermeire teaches of using memory addresses or pointers in the runtime framework as a means to access data, (Vermeire, [0033]). As for the P-invoke interfacing, this is well known in the programming field and examiner asserts Official Notice to the common use of such an interfacing in this field of endeavor and point out KSR principles which provide rationale on why one of skill would use the teaching. These include well known technique, simple substitution and obvious to try given its common use and benefits.

Therefore it would be obvious to one of ordinary skill in the art at the time of the invention to integrate the use of pointers, as taught by Vermeire, with the AAPA with the motivation that it is common to use pointers in the art as well as for better efficiency, memory management and to prevent memory leaks.

AAPA teaches in Claim 9:

The system according to claim 8, said component exposing a COM interface. **(Figure 3, [0009] shows the COM 303)**

AAPA teaches in Claim 11:

The system according to claim 8, said component including a pen services component. **(Figure 3, [0009] discloses the component 302 may include pen services)**

AAPA teaches in Claim 12:

The system according to claim 8, further comprising: at least one pen device driver sending information to said component. **(Figure 3, [0009] shows the pen device drivers 301)**

AAPA teaches in Claim 13:

The system according to claim 8, further comprising: said application including a pen input managed client. **(This is obvious in light of Figure 3's unmanaged and managed code relating to pen input from a digitizer)**

AAPA teaches in Claim 14:

A computer-readable storage medium having a program stored thereon for transferring information related to ink between an unmanaged pen input component and a managed stylus input subsystem of an application on a computing device for use by an application on the computing device **(Figure 3 shows the unmanaged and managed sections as the data is converted)**, the unmanaged pen input component being native to and executed directly by a

processor of the computing device **(Figure 3, [0009] shows the common language run-time to manage the various types of codes, such as managed and unmanaged code)**, the managed stylus input subsystem of the application being separate from the unmanaged pen input component and being executed in a common language run-time environment of a framework operating on the computing device **(Figure 3, [0009] shows the common language run-time to manage the various types of codes, such as managed and unmanaged code)**, the common language run-time environment of the framework operating on the computing device, the common language run-time environment of the framework executing the managed code independent of a type of the processor of the computing device **(Figure 3, note the managed code is a separate section of code, apart from the raw, unmanaged code of the stylus input)**, said program comprising the steps of:

receiving pen data in a pen service operating in a user mode **(Please note in Figure 3 of a kernel section being separated from 302. Programming windows often consists of a kernal mode and a user mode/windows programming mode which are distinct from each other. Examiner asserts Official Notice to this common architecture. The separated area is indeed the user mode area given the application area for user input)** on the computing device and written in unmanaged code and separate from the application on the computing device **(Figure 3, [0009] shows the component 301, on the unmanaged side)**, the pen data being generated by a digitizer of the computing device upon movement of a stylus with respect to a surface of the digitizer and being received by the pen service from a pen device driver operating in a kernel mode on the computing device, the pen data including at least one location

on the digitizer of the stylus **(Figure 3, [0009] shows the pen device drivers 301 which receives information from a digitizer. Please note that 301 is in the kernel mode)**;

the pen service transferring information related to said pen data to a shared memory on the computing device designated to be shared between unmanaged pen input component and the managed stylus input subsystem **(Figure 3, [0011] discloses the use of a memory for storing the pen information and then converting that data. Please see the combination below)**;

loading a pen input component into the application on the computing device, the pen input component being written in unmanaged code and being designated to communicate with the pen service for the application **(Figure 3, [0009] shows component 302 which is on the unmanaged side and communicates with 301 to send the information to 305)**;

the loaded pen input component further transferring to a stylus input subsystem of the application on the computing device, the stylus input subsystem being separate from the pen input component and being written in managed code and executed in the common language runtime environment **(Figure 3, [0009] show the pen component's information is retrieved by 305 which invokes a command to get the data. Figure 3, [0009] shows the application which receives the managed code after it has been converted from the unmanaged code in the CLR. This is done in the runtime callable wrapper 305 which contains a type library importer. [0009] discloses 305 and 306 import the data from the pen input component (read these as being a stylus input subsystem, which is reasonable given a subsystem is just a follow-up to the primary pen component). [0009] discloses command that can be invoked to retrieve information from pen component 302 and the callable wrapper can obviously, if not inherently, perform other actions to convert the unmanaged code)**; and

the stylus input subsystem of the application submitting with a retrieval command to the shared memory to retrieve said information from said shared memory, the retrieval command as submitted by the stylus input subsystem comprising a P-invoke method "GetData" which takes into account a context of the stylus input subsystem **(Please note [0009] and [0076] for example. Both detail marshalling code and converting code using 304, 305 and 306. It is also obvious and well known that GetData is a commonly used command in CLR and Examiner asserts Official Notice to this. In addition, it is obvious, if not inherent, that there is a command to initiate the retrieval of data in Figure 3)**; but

AAPA does not explicitly teach of a "mutual exclusion" shared memory so that non-simultaneous sharing of code can be achieved between the unmanaged and managed sections.

However, the use of mutex is well known in the art.

To emphasize, Price teaches of using a mutex to allow multiple programs to share the same resource, but not simultaneously, (Price, [0060]).

Therefore, it would be obvious to one of ordinary skill in the art at the time of the invention to integrate the mutex, as taught by Price, with the AAPA's device with the motivation that by non-simultaneously sharing the mutex, sequential operation can be done and errors can be prevented from different programs trying to access the same object at the same time.

AAPA also does not explicitly teach of a pointer to reference different parts of the memory, pen services, pen input component, etc, to either to store or retrieve them using the pointers. In addition, they are used in a P-invoke-style interface to interface between the unmanaged and managed portions.. However, both of these elements are well known in the art.

To emphasize, in the same field of endeavor, framework systems, Vermeire teaches of using memory addresses or pointers in the runtime framework as a means to access data, (Vermeire, [0033]). As for the P-invoke interfacing, this is well known in the programming field and examiner asserts Official Notice to the common use of such an interfacing in this field of endeavor and point out KSR principles which provide rationale on why one of skill would use the teaching. These include well known technique, simple substitution and obvious to try given its common use and benefits.

Therefore it would be obvious to one of ordinary skill in the art at the time of the invention to integrate the use of pointers, as taught by Vermeire, with the AAPA with the motivation that it is common to use pointers in the art as well as for better efficiency, memory management and to prevent memory leaks.

AAPA, Price and Vermeire teach in Claim 15:

The computer-readable storage medium according to claim 14, said program further comprising the steps of:

transferring additional information from said at least in part managed application to said shared memory **([0011] discloses a memory. Price has been combined with AAPA to teach of a mutex as well)**;

transferring a pointer that points to said additional information to said component; retrieving said additional information from said shared memory. **(The combination with Vermeire teaches to use pointers to access and store data in memory)**

AAPA teaches in Claim 17:

The computer-readable storage medium according to claim 14, said program further comprising the step of: exchanging information through a COM interface. **(Figure 3, [0009] shows the COM 303)**

AAPA teaches in Claim 18:

The computer-readable storage medium according to claim 14, said component being a pen services component. **(Figure 3, [0009] discloses the component 302 may include pen services)**

AAPA teaches in Claim 19:

The computer-readable storage medium according to claim 14, said application including a pen input managed client. **(This is obvious in light of Figure 3's unmanaged and managed code relating to pen input from a digitizer)**

AAPA teaches in Claim 20:

The computer-readable storage medium according to claim 14, said component receiving input from at least one pen device driver. **(Figure 3, [0009] shows the pen device drivers 301)**

As per Claims 3, 10 and 16:

These claims are directed to the use of a P-invoke style interface. Examiner takes Official Notice as to the use of P-invoke interfacing. This is common in the art as a software means and please also see the combination with Vermeire.

Response to Arguments

4. Applicant's arguments considered, but are respectfully not persuasive.

Applicant's representative is thanked for the interview to discuss the invention to discuss the differences between the prior art of Figure 3 and the invention of Figure 4. It is appreciated that repeated efforts are being made to push the case closer to allowance. Thank you.

Some of Applicant's arguments are with respect to the unmanaged and managed code, but there are no claim amendments on this. This has been discussed several times during interviews and examiner feels that the prior art clearly teaches of unmanaged and managed code sections and that more differences between the particulars of each must be better claimed.

This set of amendments deals mostly with trying to differentiate the modes of codes. However, in Windows architecture, it is well known that there are different modes, such as a kernal mode, a user/programmable mode, etc. Applicant's own disclosure, and in the background, make note of how the user interacts via the pen component 302/402 and the input

from the application 307/407. This is not using Applicant's invention against him, just pointing out the well known teachings. It is obvious that the kernel mode is separated out as shown in Figure 3 and that the other section is another mode, the user mode. If Applicant disagrees with this interpretation, examiner will be happy to provide references which reinforce this obvious, if not inherent, teaching. Respectfully, this is obvious.

Applicant has also not argued how or why the prior art would not teach of this. It seems to examiner that the differences in the prior art Figure 3 and the invention Figure 4 is not in the user modes, but rather that area as the unmanaged code is converted into managed code. Applicant's summary points out that this is the inventive concept and makes no mention of the user mode in Figure 4, further enforcing examiner's position that it is indeed shown in Figure 3.

Furthermore, simply claiming a user mode with no corresponding language is broad. It would be obvious to one of ordinary skill in the art, given the well known knowledge about the few operating modes in Windows architecture, that this section is indeed the user mode.

Applicant is advised to overcome the current rejection by better claiming the differences between Figure 3 and Figure 4. For example, how does the shared memory communications 404 factor into the invention and its uses? Figure 3 does not seem to show such this. Respectfully, Applicant is asked to perhaps claim this to overcome the current rejection.

Conclusions

5. Applicant's amendments and non-persuasive arguments necessitated the new ground(s) of rejection presented in this Office action. Accordingly, **THIS ACTION IS MADE FINAL**.

See MPEP 706.07(a). Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a).

Any inquiry concerning this communication or earlier communications from the examiner should be directed to DENNIS P. JOSEPH whose telephone number is (571)270-1459. The examiner can normally be reached on Monday-Friday, 8am-5pm.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Amr Awad can be reached on 571-272-7764. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

DJ

/Amr Awad/
Supervisory Patent Examiner, Art Unit 2629